软件工程5要素：项目、人、过程、方法和工具、软件制品

传统软件生存周期：可行性分析、需求、设计、编码和调试、软件测试、运行维护（退役）（7步）

# 1、软件危机

## 什么是软件危机

表现：60年代末，计算机应用较为普遍，软件数量大大扩大，软件复杂度超高

- 非正规软件开发方式对大型、复杂的计算机应用已力不从心，不能奏效、软件质量差、进度一拖再拖的现象普遍

## 原因（6个）

"开发人员需求固有缺陷" 2 1 2 1

2需求：用户对软件需求的描述经常出现二义性、不确定性、遗漏、或错误。开发过程中变更需求

开发人员的对用户需求的理解与用户本来的愿望有差异

项目管理人员～设计人员和开发人员：大型软件项目中各类人员的信息交流不及时、不准确、有时会产生误解

开发人员不能有效、独立自主的处理大型软件的全部关系和各个分支、易产生疏漏和错误

缺乏有力的方法学和工具支持，过分的依靠程序设计人员在软件开发过程中的技巧和创造性、加剧软件产品的个性化

软件产品的特殊性和人类智力的局限性

# 2、软件过程模型（4个）

## 考试内容

- 各模型、模型的主要特点、需要一定的总结、优缺点描述

## 瀑布模型 又叫软件生存周期模型

- 特点（3个）
  1. 将软件开发过程分成7个阶段，既是软件开发过程的分解，也是软件生存周期的阶段划分
     - 可行性研究
     - 软件需求
     - 设计
     - 编码与调试
     - 测试
     - 运行与维护
     - 退役
  2. 思路简洁、明确，上一阶段的开发结果是下一阶段开发的输入，相邻的两个阶段具有因果关系、紧密联系。
  3. 每一阶段都有阶段性制品（文档、原型、程序）
  4. 对于规模较小、软件需求比较稳定的项目或子系统采用瀑布模型能够比较显著提高开发的质量和效率
  5. 适应单主机计算模式下的软件开发过程、是软件工程技术开发和项目管理的基础

优点：有利于结构设计，规范了软件开发

- 缺点（3个）
  1. 必须要求客户和系统分析员确定软件需求后才能进行后续软件开发工作
  2. 需求确定后，用户和软件项目负责人需要等相当长的时间，才能得到一份软件的最初版本，无法及时得到反馈

如果用户对这个软件提出比较大的修改意见，那么整个项目将蒙受巨大的人力、财力和时间方面的损失

无法适应软件需求的动态变更

上游过失误导下游开发

③ 开发人员在瀑布模型上游出现的过失回味软件制品带来缺陷，会误导下游的开发活动，若未被发现，软件运行的时候或造成系统故障。

- 注意
  - 明确需求后再施工
  - 每个阶段的结果要及时评审和测试，发现的问题妥善处理后再开始下一阶段的工作
  - 开发周期尽量短、尽快给用户使用、得到反馈意见
  - 一次开发的软件规模不宜过大

针对瀑布模型开发时间长，不能快速及时得到反馈的缺点

## 增量过程模型

按照需求划分

将把握好需求分解，划分为多到增量，并为增量排序，急需的增量放在前面，稳需的放在后面，每个增量都历经需求、设计等阶段

需求分解、按需求程度安排开发顺序。每个阶段按瀑布开发

- 优点(2个)
  ① 开发过程中持续不断地发布软件新版本以便获得客户反馈，用于调整后续的软件开发策略
  ② 由于软件开发需求是确定的，可先对软件体系结构进行设计，使增量开发过程能保持良好的软件体系结构
- 缺点(3个)
  ① 增量规模不宜过大，否则会暴露瀑布的缺点（每个增量是按照瀑布的模式开发的）
  ② 将客户需求分解成增量序列必须对系统需求十分了解，并有顶层设计的经验
  ③ 多数系统都需要基本服务、如何为基本服务定义增量、何时实现这些增量、处理起来比较困难。

多批向用户提交产品，每批提交一个满足用户需求模块的可运行产品。

## 原型建造模型 → 既是迭代模型，又是进化模型

- 特点
  根据客户提出的软件定义，快速开发一个原型，向客户展示功能，在征求意见的过程中修改与完善软件。

根据需求

部分功能

- 途径(3种)
  ① 模拟软件(用模拟计算机来模拟软件系统的人机界面和人机交互方式)
  ② 用敏捷软件开发方法开发一个原型
  ③ 找来若干个相似的软件，用其展示软件需求中的部分或全部功能
- 原型种类(2种)
  ① 抛弃型原型，利用它定义和确认了软件需求后，它就完成任务
  ② 应用型原型 → 又称进化型原型，利用它确认需求后，对原型进一步加工完善，使之成为系统的一部分。

## 螺旋模型

从螺旋上升 → 是个迭代模型，也是进化模型

- 每个回路的过程
  - 确定目标
  - 风险分析
  - 开发与验证
  - 制定计划
- 优点
  - 适合大型软件开发
  - 需求可以不用完全确定
  - 可以边学习、边建模、边开发、边使用
- 缺点
  - 对需求的不确定性
  - 初期无法进行软件体系结构设计，多次迭代会导致体系结构变坏

## 需求分析 → 最重要的技术文档：需求规格说明书

输入制品中，用例模型最重要，输出制品主要是分析模型

## 3、★需求分析的过程模型

① 需求优先级分析
② 用例分析
③ 分析模型评审
④ 为辅助需求分析而构建快速原型

### ① 需求优先级分析

主要任务：为每项需求确定优先级 → 确定对谁的什么

有分析需求优先级的：
- 基于实现紧迫度的优先级
- 基于产品可接受度的优先级
- 基于需求实现的优先级
- 优先级=价值/（成本*成本权值*+风险风险权值）→ 一般而言，优先级取决于外因素的综合作用

### ②★★用例分析

目标：将用例模型中对用例的自然语言描述转化为更接近于软件设计的分析模型。

用例分析的活动：
① 精化领域概念模型
② 设置分析类
③ 构思分析类之间的协作关系
④ 导出分析类图

- 精化领域概念模型
  - 为用例分析奠定概念基础
  - 提取跨越多个用例的领域，建立公共概念。
- 设置分析类 → 与待开发软件的具体实现无关
  - 边界类<> <<boundary>>
    - 负责因为软件系统与外部执行者之间的交互
    - 界面控制
      - 输入转换、输出呈现、界面切换
    - 外部接口
      - 目标系统与外界的信息交互与互操作
    - 环境隔离
      - 与系统别的部分尽量独立
  - 控制类<> <<control>>
    - 完成用例任务的责任承担者，负责协调、控制其他类共同完成用例规定的功能或行为（通常都处理具体的任务细节）
  - 实体类<> <<entity>>
    - 负责保存目标软件系统中具有持久意义的信息项，并向其他类提供信息访问操作
- 构思分析类之间的协作关系 → 任务：将用例模型中的用例描述转为UML交互图。
  - 主动执行者向边界类发出命令
  - 边界类仅仅完成必要的输入信息解析，将命令传给控制类
  - 控制类可向实体读/写信息
  - 控制类将结果通知边界类
  - 边界类结果呈现
- 导出分析类图 → 根据每个分析类的职责来确立该分析类的属性
  - 创建初始分析类图
  - 确定分析类的职责
  - 确定分析类之间的连接
  - 根据交互图确定分析类的属性
  - 整理

### ③分析模型评审

软件相关方
做：需求工程师、软件设计师、项目软件经理、利益相关方的代表和业务专家
一旦正式地审查结束为止需求分析阶段的所有工作成果。

### ④为辅助需求分析而构建快速原型

## 5、★软件设计基本原则

① 抽象与逐步求精
② 模块化
③ 信息隐藏
④ 关注点分离

抽象而逐步求精

模块化

- 模块分解（今年不考这个） 从低到高
- 内聚度（需要再听一下） 偶然性~、逻辑性内聚、时间性~、过程性~、通信性~、顺序性内聚、功能性内聚
- 耦合度（从低到高）
  - 非直接耦合
    - 两模块模块都不依赖另一个
  - 数据耦合和控制耦合
    - 两模块通过参数交换信息、并且这些信息进餐与计算而不影响模块的功能或执行路径
  - 外部耦合
    - 与外部设备、环境相关联
  - 公共耦合
    - 模块通过公共的数据环境相互作用
  - 内容耦合
    - 两模块的业务逻辑处理线索相互交织

6、☆ **强内聚、松耦合原则** 就、就、好处

- 强内聚
  - 含义：一个软件模块由逻辑相关性恒强的代码组成，仅负责单项职责。
- 松耦合
  - 含义：软件系统中各模块尽可能独立地完成各自的职责，模块之间的接口尽可能少而且简单
- 好处 强内聚、松耦合
  - ①有利于促进软件系统的结构简单化、清晰化，从而强化设计模型的简单性
  - ②强内聚有利于软件模块的可复用性
  - ③松耦合有利于软件系统的可修改性和可维护性
  - ④更好的支持模块的并行开发，前者保持每个模块的功能相对单纯，后者在编码、调试、测试阶段可以减少模块之间相互制约的情况
- 如何实现
  - 尽可能每个模块的职责尽可能简单并且自然
  - 当有多种方案可供选择时、设计师应以强内聚、松耦合作为设计决策的主要依据

**信息隐藏（~~主观~~）** 要求模块设计得使其所包含信息（过程和数据）对于那些不需要这些信息的模块不可访问，模块间仅交换那些为完成系统功能必须交换的信息

- 强化模块之间的独立性
- 支持并行开发
- 降低测试和后期维护的工作量
- 降低修改影响向外传播的可能性

**关注点分离（~~待考证~~）** 指问题求解者针对概念、任务或目标的某个部分或侧面的聚焦。

**体系结构精化** → 主要目标：将概念级结构的描述成为全面的、设计质量的软件体系结构。

总结下7、☆
①搜索并选取可用的设计资产
②设计技术支撑设施
③确立设计元素
④整合设计元素

**逻辑视图体系结构精化**

- 搜索并选取可用的设计资产 → ①当前项目直接可供复用或借鉴的设计资产
  - 用别的系统中现成的模块、或具有重构价值的 ②由现成的系统开发机构确立的构件库、服务库
- 设计技术支撑设施 ③业界广泛采用的类库中间件或框架
  - 数据持久存储服务 ④Internet上的开源软件
  - 安全控制服务

名字完全的基础性软件技术问题，为应用功能提供服务。

实现一组技术支撑机制

- 说白了就是设计往系统里丢插件（数据库、安全框架，或者别的啥）
- 确立设计元素 → 以概念体系结构的模块为基础，以软件需求的实现为目标，探索如何将概念模块组织为设计元素，进一步研究这些设计元素之间的职责如何划分，它们之间如何协同工作。
  - 确定子系统及其接口
    - 确定的方法
      - 用例分组，每组为一个子系统
      - 将具有相关、相似职责的控制类归为一个子系统
      - 实体类分组，每一组为一个子系统
      - 与外界设备、系统交互，只需定义面向软件的接口即可
    - 评估标准
      - 较好的相关性与相似性
      - 因负责相对独立的一组处理功能
      - 两个子系统不应该具有相似性
      - 子系统规模均衡
      - 接口极小化，隐藏内部细节
  - 确定关键设计类
- 整合设计元素 整合所有设计元素
  - 整合可复用的既有设计元素、技术支撑设施中的设计元素、概念体系结构和需求模型中推导出来的设计元素（子系统），以协同完成所有的软件需求，最终给出软件体系结构的完整视图。

# 软件测试

## 软件测试的概念

- 软件测试的任务
  - 找错 运行程序或模拟程序运行，发现程序缺陷。

## 测试阶段的信息流程

- 收集、利用软件运行失误和测试数据可构造软件可靠性模型，据此数据预测出软件的可靠性

## 测试用例及其设计

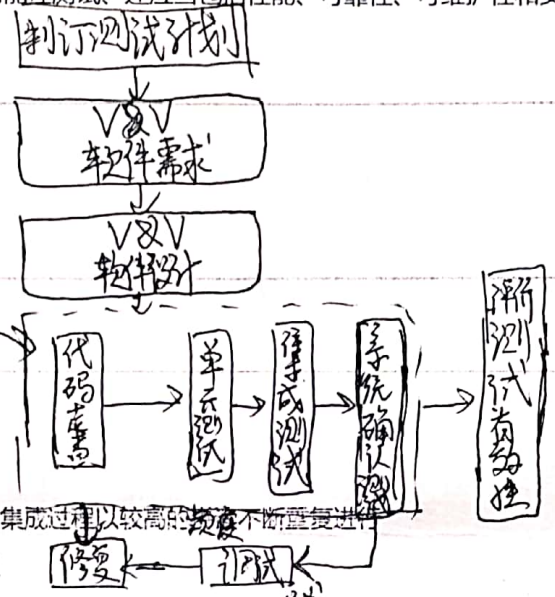
- 黑盒测试 → 按照产品的设计目标，测试产品应具备的功能，检验产品能否在使用环境中正常工作，并提供产品应具备的功能。
  - 按照产品工作原理和过程，测试产品内部各个子系统或部件的功能、属性、动作是否正常，如果正常则满足质量标准
  - 旨在说明 不考虑程序的内部结构和处理过程，通过程序的运行验证测试数据的处理结果是否与期望一致
    - 软件的功能是否可操作
    - 软件是否能适当的接收输入数据并产生正确的输出结果、或在可能的场景中事件驱动的效果是否尽如人意
    - 能否保持外部信息（如数据文件）的完整性
- 白盒测试 → 关注程序的运行细节，对对程序的每条逻辑路径分别设计测试用例，检查分支和循环的执行情况。
  - ~~穷举所有的逻辑路径~~
  - 实践中，一般根据程序路径的重要性、存在缺陷的可能性、路径运行的频率等选用测试用例，达到尽可能少的资源，取得最有效的测试效果
- PS
  - 应当是实验题类型、不太可能简答题、因此需要掌握方法

## 软件测试的原则

- 测试是一个持续的过程，而不是一个阶段
- 测试一定有计划、受控制、并提供足够的时间和资源
- 测试应当分优先级
- 测试应当有重点
- 测试不是为了证明程序的正确性，而是为了证明程序不能工作
- 测试是不可能穷尽的，当测试充分性满足时就可以停止测试
- 测试是开发的朋友
- 测试人员应公正地测试、如实地记录和报告缺陷
- 测试自动化能解决一部分问题，但不是全部
- 测试不能仅仅包括功能性测试、还应当包括性能、可靠性、可维护性和安全性等方向的验证

## 软件测试过程

- 软件测试过程.png




## 持续集成

### 概述

- 什么是
  - 相关概念
    - 持续是指集成过程以较高的频度不断重复进行
    - 狭义集成
      - 通常包括代码之间的集成、以及集成后代码的编译、单元测试与集成测试
    - 广义集成
      - 在此基础上还可选地包含代码质量分析、产品打包与发布等动作
- 为什么要
  - 大型软件项目的开发周期较长，并且在集成时极易发生错误
    - 模块之间的接口不匹配
    - 集成后跨模块的流程逻辑出现异常
    - 多模块使用的统一库程序的版本冲突
  - 如果集成频度过低，那么错误被隐藏的时间就会太久，导致改正错误的代价剧增、软件产品延迟交付的风险越大
- 怎么做
  - 过程（7步）
    - 构建
    - 单元测试
    - 集成测试
    - 确认测试
    - 代码质量分析
    - 发布
    - 部署
  - 典型的集成场景描述
    - 提交代码
      - 新代码->VCS

- 触发集成
  - 按照预先配置的“推”或“拉”方式触发管道中首个动作
- 执行集成
  - 需要脚本程序
- 发送通知
  - 可以对主、从工具进行配置以支持想代码提交者发送集成过程中的进展即错误概述信息

## 版本控制

- 直接扯git

## 构建（大致描述）（直接扯打包Maven或者Graddle）

- 编译源代码并将编译生成的多个目标代码文件封装为单个（或少数几个）可执行文件的过程
- 典型的构建操作
  - 编译
  - 复制或移动至指定目录
  - 封包将该目录下多个目标码文件封包

## 测试

- 灵活高效地实现测试自动化
  - 单元测试自动化
    - 针对单个类展开测试
  - 集成测试自动化
    - 多模块集成起来测试是否能够协调工作
  - 确认测试自动化
    - 站在用户视角来发现软件系统的缺陷

## 代码质量测试

- 借助代码质量分析工具实现自动化分析
  - FindBugs
  - PMD
  - CheckStyle
  - HP Fortify

## 发布与部署

部署工作的内容一般包含检测硬件和软件基础环境是否满足要求，必要时升级基础软件版本，停止旧版运行，清理环境中无用的旧版文件，从作品库提取发布包并传递至目标环境，启动新版软件并执行冒烟测试。

- 软件发布
  - 将可执行文件（有时还包括部分源代码文件）和必要的说明文档（使用说明等）使用打包工具制作成软件包，然后将软件包传递至中心代码库或直接提交给使用者。
- 持续集成的目标
  - 确保软件版本的每次重要更新都可以敏捷地部署至工作环境
- 冒烟测试
  - 在生产环境运行前需要在虚拟环境中进行冒烟测试

- 冒烟测试的对象是每一个新编译的需要正式测试的软件版本，目的是确认软件基本功能正常，可以进行后续的正式测试工作。
- 冒烟测试的执行者是版本编译人员。

# 11. 风险分析

识别、估算、管理评价

## 风险标识

- 项目风险
  - 项目在预算、进度、人力、资源、顾客、和需求等方面原因对项目产生不良影响
- 技术风险
  - 软件在设计、实现、接口、验证和维护过程中可能发生潜在的问题（规约的二义性、采用陈旧或不成熟的技术）
- 商业风险
  - 开发了一个没人需要的优质软件、或者销售部门不知道如何推销这一软件产品，或者开发的产品不符合公司的产品销售战略等

## 风险评价和管理

- 风险估算（风险的发生概率、严重程度）
  - 在风险分析过程中，经常使用三元组[Ri,Li,Xi]描述风险。其中Ri代表风险;Li表示风险发生的概率;Xi是风险带来的影响，I = 1，2，...L是风险序
  - 估算过程
    - ①定义项目的风险参考量;
    - ②定义每种风险的三元组[RI,LI,XI];
    - ③定义被迫终止的临界点
    - ④预测几种风险组合对参考值的综合影响
- 风险管理
  - 建RMMM计划 (Risk Mitigation Monitoring Management)
    - 缓解
    - 监测
    - 管理
  - 用风险信息表RIS

## 如何避免

- 项目开始前因从源头上避免产生风险的要素，开始后想方设法减轻风险要素的影响
- 了解项目开发人员变动的原因，项目开发期间应采用必要措施尽量减少人员流动
- 在工作方法和技术上因采用适当的措施、防止人员流动给工作带来损失
- 项目在开发过程中应及时公布并交流项目开发的信息
- 建立组织机构，确定文档标准，并按要求及时生成文档
- 经常对工作进行集体复审，并按要求及时生成文档
- 为关键技术培养、准备后备人员

☆UML定义了5类图形机制.

用例视图{用例图

结构视图{类图
包图
对象图
组件图
部署图

行为视图{交互图{顺序图
通信图
状态图
活动图

构件视图{构件图

部署视图{部署图

静态建模机制{类图
包图

动态建模机制{顺序图
通信图
活动图
状态图

用例图:

用例:执行者与目标系统的一次交互.

相对独立性和通性是用例必备的两项特征.

用例是功能性需求的主体部分

☆用例图:

节点:节点有两种{执行者  "叫我十八"
用例

边:节点之间的边表示两个节点间的关系.

执行者和用例间通常为无向边,有向边并不完全代表信息传递方向

用例间的关系主要有3种:包含(include)、扩展(extend)、继承(inheritance)

<<include>> <<extend>> ⟶

执行者之间的继承关系:⟶

布局:①注重执行者在左上角

②主动执行者在用例左侧,被动执行者在用例右侧

③多个用例垂直排列

④在水平方向绘制包含,被包含的用例放在右侧

⑤在垂直方向绘制扩展和继承,被扩展/继承的用例放上方

类图

节点：描述系统中的类及其属性和操作

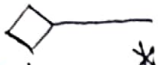
边：边表示类之间的联系


+ public
- private
~ default
# protected
_(例)_ static
斜体 abstract

类之间的关系：

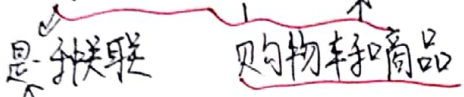
继承：

聚合：



部件类对象可能是整体类对象的组成部分

关联 { 是一种关联



有箭头(有方)的地方表示

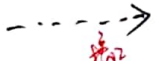
组合：

一个部件类对象只能位于一个整体类对象中

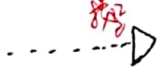
一旦整体类对象消亡，其包含的部件类对象也不能存活。

表达，较硬性关联

有向 { 依赖：有向，A依赖B，B的变化将致A的变化

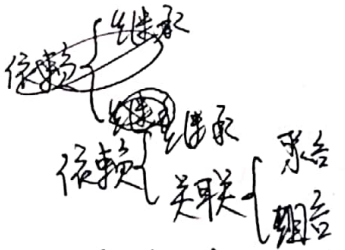


实现：



表达，是一种特殊的依赖

有向是高线




耦合度：继承 > 组合 > 聚合 > 普通关联 > 依赖

布局：① 继承尽量垂直，其他尽量水平

父类在子类上方，有向关系尽量从左到右

整体类一般在部件类左侧。

② 多条相同关系的边可以合在一块

# 活动图

描述实体为完成某项功能而执行的操作序列。

活动图包括控制流和信息流

节点(4种)：

开始节点：●

活动：⬭

终止节点：◉

决策点：◇

并发控制

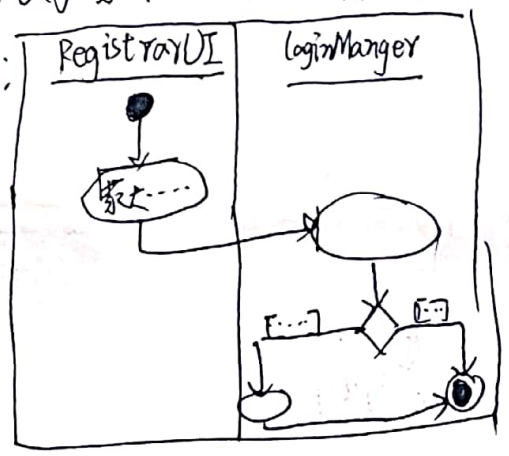
对象：

边(两种)：→

控制流：有向边，两个非对象节点之间

对象流：对象节点指向活动节点

泳道机制：



## 顺序图：

也叫序列图，和通信图语义相同。

纵向向代表时间轴，时间从上到下延伸。

横向由参与交互的对象构成，对象间无顺序关系

图形元素：对象及其生命线、活动期、消息传递、注解

① 对象：矩形框内 [对象名:类名]

② 生命线：垂直虚线

③ 执行模块(活动期)：生命线上的矩形框

④ 消息传递：类对象生命线之间的有向边

边上可标注 [监护条件] 返回值:=消息名(参数表)

消息分同步和异步两种。

同步：实心三角形箭头 →

异步：普通箭头 →

4种特殊消息：自消息、返回消息、创建消息、销毁消息

返回消息：虚线有向边

<<create>>

<<destroy>>

布局：

① 主动执行者在左侧，被动执行者在右侧。

② 主动执行者 — 边界类 — 控制类 — 实体类 — 边界类 — 被动执行者。

边界类 → 外部接口和环境隔离层

还有个功能是外部接口

UML扩充机制 "构约标"
约束、标记值、构造型等

用例设计、子系统设计、构件设计、类设计
数据模型设计、设计类验证

数据模型设计一般是在详细设计阶段开展的.

数据流图包括实体、自上而下逐层分解
是结构化分析的一种工具

传统数据流主要由4块:数据源
数据流、控制流、数据存储组成.
5部分

软件规模度量方法主要有3种
实现 设计 需求
代码行度量 功能点 对象点度量
依赖语言
不依赖于语言
能比较使用不同语言编写的程序的规模

软件质量要素(11个,3类)
第一类要素(软件的运行特征)
正确性、可靠性、有效性、完整性、可用性 "效确靠用整"

CMM能力成熟度等级
L1初始级、L2可重复级、L3定义级、L4已管理级、L5优化级

用户界面设计模型 → 表示:屏幕内容的表示、屏幕之间是如何转移的表示
4种元素:静态元素、动态元素、用户输入元素、用户命令元素
UML类:省略 类属性 属性或类型 类操作

任何一个关键建活动力超前完成,都将促进整个工程提前完成X

软件维护
纠错性维护
适应性维护
完善性维护
预防性维护

软件周期其间中占用精力和费用最多的阶段往往是
运行和维护阶段

"完善适应"
"预防纠错"
"预防纠错完善"

→或需求模型.

在软件开发过程中，需求规格说明书是软件设计的基础，也是软件验收的依据。

由继承导致的耦合度高于除其之外的其他关联关系 ✗ 继承>组合>聚合>关联>依赖

编程风格很大程度上影响程序的可读性，可测试性和可维护性。✗

在软件需求分析的过程中，可以尽早设计测试用例 ✓

程序的可读性不仅影响软件维护成本，还影响正确性和可靠性。✓

数据流图和UML都是一种建模语言，可用于对软件需求进行描述

软件测试的目的：发现软件中的故障/隐藏的错误

软件风险不一定会造成负面影响

在软件设计过程中，采用信息隐藏原则有助于提高软件产品的可修改性

软件需求规格说明书的质量要素主要有正确性、无歧义性、完全性和可验证性、一致性等（"正确完整一致"）

面向对象的软件设计模型主要包括：体系结构模型、用户界面模型、用例设计模型和数据模型等。（"倒体界数"886中）

UML类图中，耦合强度：继承关系>组成关系>聚合关系>关联关系>依赖关系（没有说实现）

黑盒测试的主要方法有等价类法、边界值分析法、又称测试法

软件项目管理的主要依据有到底项目实施计划、对人员进行组织分工等

在进行软件项目策划时，即使设计结果不精确，软件设计仍然很有必要。✓

在进行软件测试时，若多次白盒测试未出错，则黑盒也好出错 ✗

一般，在一个软件系统中，模块内聚度越强，模块之间耦合度越弱。✓

在软件需求规格说明书中，必须详细说明软件非功能性需求的实现方案 ✗

软件设计模型的质量要素主要有正确性、完备性、优化性和简单性等。（"简正完优"）

软件需求类别的主要内容包括系统描述、功能性需求、质量需求和约束性需求等（"功质约系"）

一条设计模式通常包括设计模式的名称、问题、动机、适用性、解决方案和效果等主要部分（"名问动适效"）

手机代码 关联模式

☆等价类划分法　闰年：能被4整除，不能被100整除，闰年2月有29天

12. 定义：根据输入数据的类型和程序的功能说明划分等价类，然后为每一个等价类设计一个测试用例。

步骤：①划分等价类（范围、条件值、条件等）

②设计测试用例

格式（例）：序号　测试用例描述、输入参数、期望输出